FACULTAD DE
INFORMÁTICA DE BARCELONA

TRABAJO DE FINAL DE GRADO

ALGORITMO DE SIMPLIFICACIÓN DE MALLAS PARA IMPRESIÓN 3D

David Bosch Serra

Director: Antonio Chica Calaf
Especialidad: Computación

Octubre 2017

RESUMEN

Las impresoras 3D se empezaron a usar hace bastantes años, pero con el abaratamiento de los precios esta tecnología está disponible cada vez para un mayor rango de consumidores. El sector que nos interesa es el de los aficionados, ya que no necesitan mucha precisión en los resultados, con que la apariencia sea la deseada ya suele ser suficiente y, además, la cantidad de dinero que pueden invertir en su hobby es bastante limitada, repercutiendo en las especificaciones del hardware que poseen. Esto dificulta, o imposibilita en según qué casos, el procesado de modelos excesivamente detallados los cuales, por otro lado, perderán detalle al ser impresos con hardware de bajo coste.

Este proyecto tiene como meta diseñar e implementar un algoritmo que permita la simplificación de modelos 3D, utilizando las características de la impresora para reducir el nivel de detalle, sin que esto repercuta en el resultado final. De esta forma, reduciendo la complejidad del modelo, se facilitará su posterior procesado e impresión. Además, también queremos proporcionar una herramienta con la que se pueda comparar el resultado del algoritmo contra el modelo inicial.

RESUMEN

Les impressores 3D es van començar a fer servir fa bastants anys, però amb l'abaratiment dels preus aquesta tecnologia està disponible cada vegada per a un major rang de consumidors. El sector que ens interessa és el dels aficionats, ja que no necessiten molta precisió en els resultats, amb que l'aparença sigui la desitjada ja sol ser suficient i, a més, la quantitat de diners que poden invertir en el seu hobby és força limitada, repercutint en les especificacions del hardware que posseeixen. Això dificulta, o impossibilita en segons quins casos, el processat de models excessivament detallats els quals, d'altra banda, perdran detall en ser impresos amb hardware de baix cost.

Aquest projecte té com a meta dissenyar i implementar un algoritme que permeti la simplificació de models 3D, utilitzant les característiques de la impressora per reduir el nivell de detall, sense que això repercuteixi en el resultat final. D'aquesta manera, reduint la complexitat del model, es facilitarà el posterior processat i impressió. A més, també volem proporcionar una eina amb la qual es pugui comparar el resultat de l'algorisme contra el model inicial.

ABSTRACT

We have been using 3D printers for many year, but now that the price is getting lower this technology is available to a wider range of users. We are aiming to the amateur consumers since they do not need much precision in the results, they are satisfied with the same appearance, in addition, the amount of money that they can invest in a hobby is quite limited, which has repercussions on the specification of the hardware they own. This makes it difficult, or even impossible in some cases, to process excessively detailed models which, on the other hand, will lose detail when printed with low-cost hardware.

This project aims to design and implement an algorithm that allows the simplification of 3D models, using the traits of the printer to reduce the level of detail, without impacting negatively in the final result. Thus, decreasing the complexity of the model, will facilitate its subsequent processing and printing. In addition, we also want to provide a tool which will allow the user to compare the algorithm's result against the initial model.

AGRADECIMIENTOS

Quiero agradecer a Antonio Chica, director del proyecto, por haberme ayudado y por haber tenido tanta paciencia durante todo este tiempo, sin él seguramente nunca lo hubiera terminado. Muchísimas gracias. Ojalá algún día te lo pueda devolver.

También agradecer a mí familia y amigos, por apoyarme y empujarme hacia delante cuando me hacía falta.

ÍNDICE

1 INTRODUCCIÓN Y CONTEXTUALIZACIÓN

En esta sección daremos una introducción a las impresoras 3D usadas por aficionados, cómo funcionan y, sobre todo, cuáles son sus problemas y limitaciones; explicaremos cuáles de estos problemas intentamos resolver y todos los actores implicados en este proyecto.

Además, vamos dividir el proyecto en los diferentes elementos que lo componen y aportaremos información sobre cuál es el estado del arte de cada uno de ellos.

1.1 CONTEXTO Y MOTIVACIONES

Las impresoras 3D no son una cosa nueva, en 1980 ya se podían ver los primeros modelos. Sin embargo, como con la mayoría de las nuevas tecnologías, es necesario que pasen los años para que los precios bajen y estas tecnologías estén verdaderamente al alcance de la gente. Así, desde hace unos cuantos años ya es posible para un aficionado, con menos de 500 $, conseguir una impresora 3D capaz de imprimir nuestros modelos con una calidad aceptable [1].

Estas impresoras 3D baratas, que utilizan la tecnología FFF (Fused Filament Fabrication) [2], siguen el mismo procedimiento a la hora de imprimir:

- El usuario selecciona el modelo 3D que desea imprimir y un software (varía según la impresora) verifica que cumpla las condiciones necesarias para ser impreso.
- Si el modelo cumple las condiciones, el programa empieza a convertir el modelo en rebanadas (slicing). Dependiendo de la impresora y el software, se empieza a imprimir a medida que están las slices o una vez se tienen todas.
- El cabezal (nozzle) va imprimiendo las rodajas, una encima de la otra, hasta terminar el modelo.

El paso de slicing, tanto en memoria como en tiempo, depende del número de triángulos que componen el modelo 3D. Debido a esto, con modelos muy detallados, y dependiendo del software utilizado, puede llegar a ser un paso prohibitivo.

Figura 1.1: Diagrama de slicing e implesión de un modelo

Además, algunos programas tienen problemas procesando modelos muy grandes, produciendo pequeñas gotas de filamento en la superficie. Presumiblemente, esto ocurre porque el software de la impresora 3D tarda demasiado en procesar ciertas partes del modelo y el cabezal segrega demasiado material [3].

Otra característica de las impresoras 3D que hay que tener en cuenta, es que mientras que pueden llegar a tener una resolución de hasta 50 micrones en el eje XY, el diámetro del cabezal suele estar entre los 0.3 y 0.5 mm. Así pues, en la práctica, el nivel de detalle que puede alcanzar una impresión dependerá del diámetro del cabezal y del espesor de las capas.

El proyecto que desarrollaremos a continuación trata de aliviar estos problemas utilizando un algoritmo de simplificación de mallas, pero que tiene en cuenta los parámetros de la impresora (diámetro del cabezal y resolución en el eje Z) para evitar perder detalle.

1.2 ACTORES

El desarrollo de este proyecto involucra a varios actores, a continuación, los citaremos y hablaremos de ellos.

Desarrollador del proyecto: Este proyecto solo cuenta con un desarrollador, yo, que me encargaré de producir toda la documentación, de realizar la investigación necesaria, de desarrollar el código y de probarlo.

Director del proyecto: El director de este proyecto es Antonio Chica, profesor asistente del Departamento de Ciencias de la Computación de la Universidad Politécnica de Cataluña. Su rol es el de supervisar el proyecto y guiarme para que el resultado sea lo mejor posible.

Usuarios: Aficionados que posean una impresora 3D y que deseen imprimir modelos muy grandes (en cuanto a número de triángulos). También pueden estar interesados los usuarios de algún servicio de impresión 3D, ya que estos suelen exigir que no se supere un número máximo de caras.

1.3 OBJETIVOS

Los objetivos que nos hemos impuesto en la realización de este proyecto, son:

- Desarrollar una aplicación que permita al usuario simplificar un modelo 3D usando los parámetros de la impresora. Esta aplicación debe contar con una interfaz que sea fácil de usar.

- El resultado debe ser útil, esto es, la cantidad de triángulos eliminados debe ser significativa y si el modelo de entrada cumplía todas las condiciones para ser impreso, la salida también debe hacerlo.

- El modelo final impreso debe conservar la misma calidad en los detalles y la misma forma que la que tendría el original al imprimirse.

- La aplicación debe permitir al usuario comparar, mediante un visualizador 3D, las diferencias entre el modelo original y el simplificado. Para hacer esto se pintará, sobre el modelo simplificado, una textura que codifique la distancia entre las dos superficies.

- La aplicación debe permitir como entrada y salida formatos estándar, por lo menos stl.

1.4 ESTADO DEL ARTE

En esta sección vamos dividir el proyecto en los diferentes elementos que lo componen y vamos a aportar información sobre cuál es el estado del arte de cada uno de ellos.

1.4.1 SIMPLIFICACIÓN DE MODELOS 3D

La simplificación de modelos 3D (mesh decimation) consiste en reducir la cantidad de elementos que componen un modelo intentando preservar, en la medida de lo posible, la topología y forma original. Es un tema sobre el que se ha escrito mucho y existen bastantes métodos diferentes, sin embargo, el proceso generalmente se puede dividir en tres pasos [4]:

1. **Clasificación de vértices:** Se clasifican los vértices del modelo de acuerdo a su geometría y la topología local. Este paso sirve para encontrar posibles candidatos para ser eliminados.

2. **Criterio de simplificación:** Se utiliza alguna métrica para ordenar los vértices de acuerdo al error que se introduce si son eliminados. Desde hace bastantes años, la métrica más utilizada es quadric error metrics [5].

3. **Triangulación:** Después de eliminar un vértice el agujero resultante tiene que ser triangulado. Es posible que se tenga que volver a calcular la métrica para la geometría local después de cada triangulación.

A continuación, describiremos dos paquetes de software que implementan, entre otras cosas, algún método de simplificación de modelos 3D:

MeshLab: Sistema de procesamiento de mallas, open source y extensible. Está pensado como un framework para el modelado y edición 3D que permita tanto a estudiantes como a investigadores reproducir los resultados de su investigación. En lo referente a simplificación de mallas, MeshLab implementa edge collapse guiado por quadric error metric [6].

OpenMesh: Librería que implementa una estructura de datos genérica y eficiente para la representación de mallas poligonales. Está pensada para que sea fácil extender las funcionalidades base y que el usuario pueda programar cualquier filtro sin tener que preocuparse por las estructuras de datos necesarias. Además, también cuenta con un framework de simplificación de mallas mediante edge collapse y varias métricas diferentes [7].

1.4.2 Visor 3D

Un visor 3D es un programa (o parte de un programa) que es capaz de leer modelos 3D y dibujarlos para que el usuario pueda verlos y, posiblemente, interactuar con ellos. Las aplicaciones que poseen un visor 3D van desde programas de diseño CAD hasta videojuegos y la cantidad de texto que se ha escrito al respecto es enorme. Actualmente, los avances en este campo van muy ligados al desarrollo de nuevo hardware que permita visualizar un mayor número de polígonos, con los que se puede aumentar el detalle y realismo de la imagen. Sin embargo, para nuestro proyecto solo requerimos que el visor pueda dibujar un polígono y que el usuario pueda moverlo y rotarlo, algo que ya se ha hecho muchísimas veces en un montón de aplicaciones, por lo que en este aspecto no aportaremos nada nuevo.

A continuación, hablaremos sobre dos proyectos que utilizan visores 3D en una manera que nos interesa para este proyecto.

Metro: Aplicación que compara las diferencias entre dos superficies (modelos 3D). Los resultados los enseña visualmente, coloreando la superficie del modelo con el error (diferencia) aproximado [8]. Esta aplicación es un buen ejemplo lo que queremos que nuestro visor pueda hacer.

libQGLViewer: Librería en C++ basada en Qt que facilita la creación de visores 3D con funcionalidades típicas como puede ser mover la cámara con el mouse. Está pensada tanto para hacer cosas simples, usando las clases ya hechas, como para cosas más complejas, modificando y extendiendo las clases base [9].

1.4.3 Comparación de superficies

Una vez simplificado el modelo, queremos que la aplicación compare las dos superficies. Para hacer esto usaremos la malla original y por cada vértice buscaremos su distancia mínima a la malla resultante (simplificada).

Metro [9], del cual ya hemos hablado antes, hace una cosa similar a la hora de comparar, la diferencia es que en lugar de buscar la distancia desde los vértices elige los puntos haciendo un muestreo aleatorio de la superficie. El resultado en los dos casos, con un nivel suficiente de subdivisión/muestreo, será muy similar, sobre todo cuando la deformación es muy pequeña, como es en nuestro caso.

1.4.4 CONCLUSIONES

Como hemos visto, algunos de los elementos que componen nuestro proyecto no son precisamente innovadores, en esta sección hablaremos sobre los que lo diferencian de lo visto hasta ahora, y debido a los cuales tiene sentido este trabajo.

En este proyecto hemos optado por usar OpenMesh para realizar la simplificación (edge collapse) y quadric error metric [6] para guiarla. La particularidad es que utilizaremos una voxelización donde el tamaño de los voxels será el diámetro del cabezal para el eje XY y el espesor de las capas para el eje Z. Usando esta voxelización evitaremos que el algoritmo contraiga aristas si esto provoca que la superficie del modelo ocupe un conjunto de voxels diferente al original, preservando los detalles que se podrán apreciar al ser impresos mientras se disminuye la complejidad del modelo.

Para el visor 3D, a pesar de que hay librerías como libQGLViewer, lo implementaremos nosotros usando instrucciones de OpenGL. Aparte de las funcionalidades base, tendremos que calcular las distancias (entre los dos modelos) y pintarlas.

Si bien, poder comparar dos mallas gráficamente no es algo nuevo, el hecho de poder hacerlo en la misma aplicación en la que se realiza la simplificación y poder comparar los resultados de diferentes parámetros aporta valor al proyecto.

1.5 ANÁLISIS DE LAS ALTERNATIVAS

Este proyecto trata de resolver dos problemas diferentes: el primero, y principal, es simplificar (reducir la cantidad de triángulos) un modelo 3D para facilitar su impresión, intentando que el resultado después de imprimirse sea lo más similar posible al modelo original. El algoritmo que proponemos ya establece el procedimiento a seguir, por lo que probar con otros métodos de simplificación o métricas diferentes cae fuera del alcance de este proyecto. Comparar diferentes algoritmos de simplificación es un tema lo suficientemente extenso como para ser un proyecto por sí mismo.

El segundo problema que tratamos de resolver consiste en comparar o medir que tan fiel es nuestro resultado en relación al modelo original. Este problema se puede dividir en dos partes: comparar las dos mallas y presentar el resultado al usuario.

Para medir el error introducido al simplificar el modelo, la mejor forma de hacerlo es calculando la distancia desde cada punto de la superficie simplificada a la original, viceversa y a partir de eso obtener la distancia de Hausdorff [10][11], el problema es que esto es imposible de hacer, por lo que necesitamos alguna simplificación donde reduzcamos el número de puntos a un conjunto finito. Hemos considerado varias opciones:

- Realizar un muestreo aleatorio de puntos sobre la superficie del modelo de origen y calcular su distancia a la cara más cercana del modelo destino. Este es el método empleado en [8].

- Dividir cada cara (triángulo) del modelo simplificado en x triángulos iguales y calcular la distancia desde los vértices del modelo original a la cara más cercana del modelo simplificado.

Los resultados de los dos métodos, con suficientes puntos o vértices, serán muy similares, pero hemos optado por usar el segundo porque será más fácil enseñar los resultados.

Una vez obtenidos los resultados es necesario enseñárselos al usuario en una forma que sea útil y fácil de entender, para hacer esto hay varias opciones:

- Hacer un histograma de las distancias de Hausdorff obtenidas.
- Dibujar el modelo simplificado y pintándolo con un color que codifique la distancia.

Hemos escogido dibujar el modelo ya que es la opción más visual y permitirá al usuario identificar las áreas más problemáticas.

1.6 TECNOLOGÍAS

Una vez vistas las principales características del proyecto y los elementos necesarios para resolver el problema, discutiremos las tecnologías que hemos decidido utilizar para resolverlo.

OpenMesh: Utilizaremos esta librería para almacenar la geometría del modelo y realizar la simplificación, ya que nos da la flexibilidad de poder programar el algoritmo que se ejecutará durante la simplificación y nos abstrae, si queremos, de las estructuras de datos y operaciones de bajo nivel necesarias para mantenerlas.

OpenGL: El visor 3D lo programaremos utilizando esta API ya que es con la que estamos más familiarizados y esto reducirá el tiempo de desarrollo.

C++: Para programar el algoritmo y la interfaz utilizaremos este lenguaje, ya que tanto OpenMesh como OpenGL lo utilizan.

Qt: Utilizaremos este framework para diseñar y programar la interfaz. Además de que es la solución que nos han enseñado a utilizar en el grado, también es el estándar usado en toda la industria.

2 ALGORITMO DE SIMPLIFICACIÓN

El algoritmo de simplificación es la parte más importante de este proyecto y a la que hemos dedicado más tiempo y esfuerzo. Empezaremos describiendo su funcionamiento básico, para luego ir detallando cada una de sus partes.

Hemos decidido usar un algoritmo voraz. Esto quiere decir que en cada iteración elige, usando una métrica y varias pruebas, un halfedge[1] que será contraído (eliminado); el proceso continúa hasta que ya no puede eliminar ningún halfedge más.

Contraer un halfedge consiste en eliminar una arista moviendo el vértice de origen del halfedge al vértice destino y fusionándolos, eliminando así dos caras y tres aristas (en el caso general).

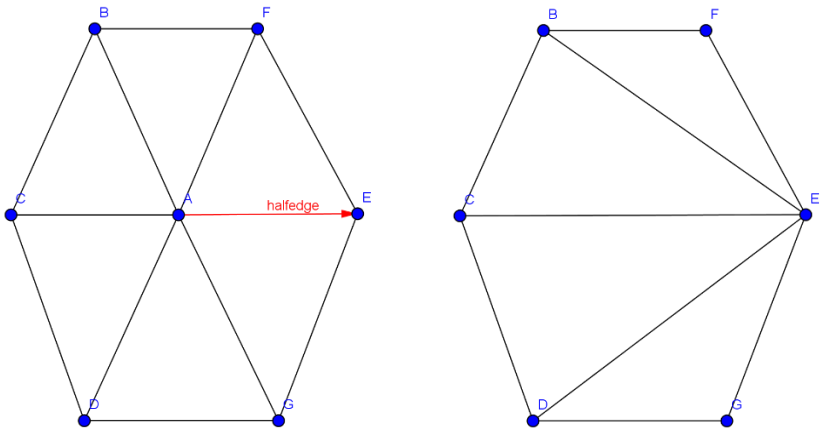

Figura 2.1: Ejemplo de halfedge collapse

Pseudocódigo del algoritmo:

```
ordenar en una cola de prioridad todas las contracciones posibles
mientras la cola no este vacía
    coger el top de la cola
    si el top cumple con todas las condiciones
        contraer el top
        actualizar la métrica de los elementos modificados
        insertarlos en la cola si no estaban
```

[1] Un halfedge es una estructura de datos utilizada para representar grafos. Una arista está compuesta por dos halfedge con orientaciones opuestas [13].

Para ordenar la cola utilizamos quadric error metrics [5], que describiremos más adelante.

Como se puede ver en el pseudocódigo, para que se realice la contracción, se tienen que cumplir todas las condiciones impuestas. Estas condiciones son:

- Ningún voxel ocupado antes de la contracción estará vacío después.
- Ningún voxel vacío antes de la contracción estará ocupado después.
- La normal[2] de ninguna cara se puede mover más de α grados.

Para comprobar las dos primeras condiciones es necesario calcular y mantener una voxelización. En la sección 2.2 daremos más detalles al respecto.

2.1 QUADRIC ERROR METRIC

Quadric error metric es una métrica que aproxima el error que se produce al unir dos vértices y lo va acumulando en el vértice que sobrevive. Desde su publicación en el 97, se ha convertido en el estándar usado para la simplificación de superficies. Esto se debe a lo eficiente que es su cálculo y lo buena que es su aproximación [5].

Afortunadamente, OpenMesh posee una implementación de esta métrica, la cual hemos decidido usar.

2.2 VOXELIZACIÓN

Una voxelización es una estructura de datos utilizada para agilizar calculo geométrico cuando la posición de los objetos es relevante. Está compuesta por una matriz tridimensional de ortoedros (generalmente cubos) regulares, o voxels, que dividen el espacio con el que se desea trabajar [14].

Según los parámetros que se utilicen al calcular la voxelización (voxels muy pequeños o un área total muy grande) es muy fácil que los requerimientos de memoria se disparen ya que el número de voxels crece cúbicamente.

En nuestro caso, en cada voxel guardamos todas las caras (triángulos) que lo intersectan o están contenidas; una cara puede estar repetida en varios voxels. El tamaño de los voxels viene determinado por los parámetros de la impresora: diámetro del cabezal × diámetro del cabezal × altura de las capas; que como ya hemos explicado es, en la práctica, la resolución de la impresora. El área de la voxelización es la caja contenedora del modelo más un margen de un voxel en todas las direcciones.

Usando una voxelización para decidir si una contracción es válida nos aseguramos que la superficie o frontera del modelo en ningún momento se mueve una distancia mayor que la diagonal de un voxel, evitando que la simplificación haga cambios en la topología si estos afectan a la apariencia del modelo después de imprimirse.

[2] La normal de una cara es un vector perpendicular a la superficie de esta, generalmente con módulo 1.

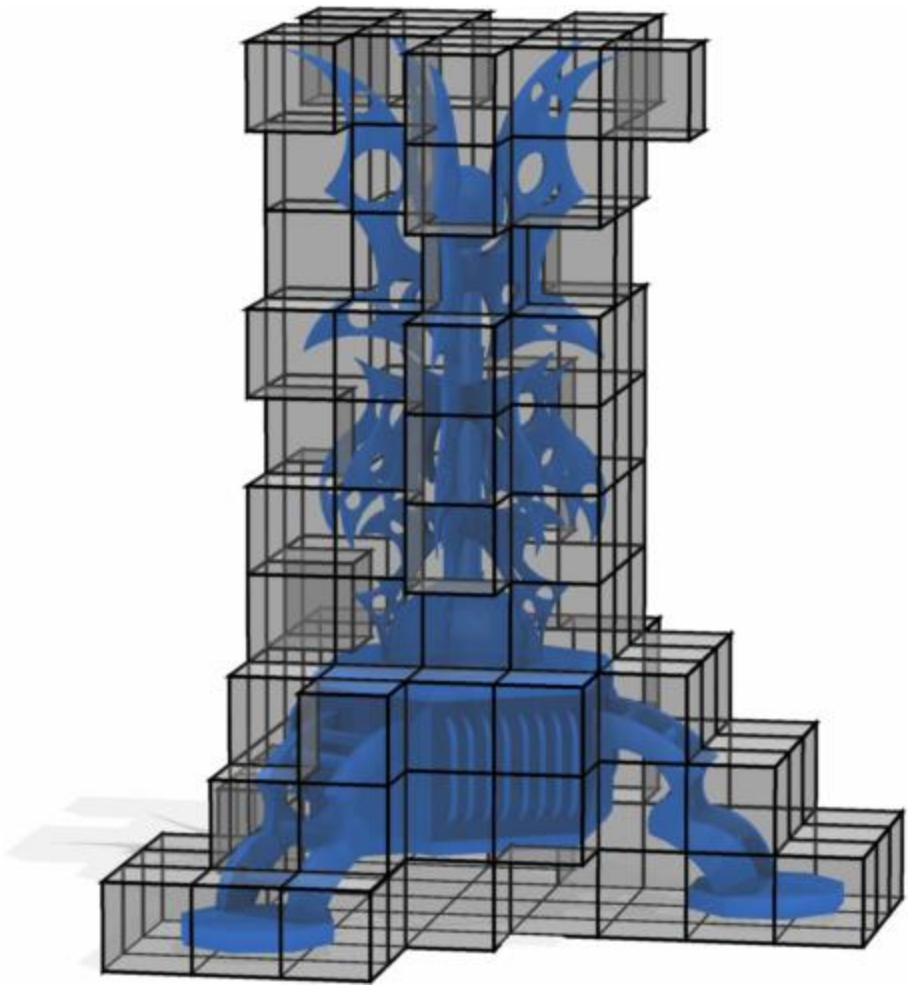

Figura 2.2: Representación de un objeto almacenado en diferentes voxels

2.2.1 Construcción y mantenimiento de la voxelización

Para calcular la voxelización se necesitan dos cosas: Dividir el área que queremos voxelizar, decidiendo el espacio que ocupará cada voxel, y calcular en que voxels guardar cada triangulo.

La primera parte, ya que todos los voxels tienen el mismo tamaño, consistirá en calcular la caja contenedora y dividir el espacio, agregando a cada voxel un porcentaje de su tamaño en todas las direcciones de forma que entre dos voxels siempre haya un espacio en el que se superponen. Esto sirve para evitar casos en los que una cara está justo en el plano que divide a dos voxels.

Para calcular en qué voxels guardar cada cara, es necesario calcular la caja contenedora del triángulo y hacer una prueba de intersección entre el triángulo y cada voxel en el que esté la caja contenedora.

Al ser, tanto los triángulos como los voxels, objetos convexos hemos decidido usar el Teorema de Separación de Ejes (SAT) [15] para decidir si hay o no intersección. El teorema afirma que si dos objetos

convexos no se penetran existe un eje para el cual la proyección de los objetos no se solapará. Además, basta con probar con las normales de las aristas.

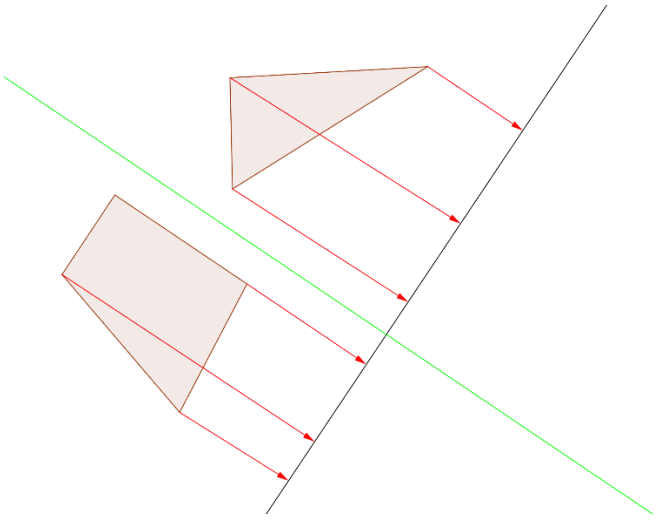

Figura 2.3: Proyección de los puntos y línea separadora (verde) de dos polígonos

Inicialmente, decidimos implementar ésta comprobación, pero después de realizar varias pruebas y ver que nuestro algoritmo era bastante lento, optamos por utilizar una implementación de este teorema que utiliza ciertas simplificaciones para hacer el código más rápido [12].

Una vez la voxelización ya se ha calculado y el algoritmo ha empezado a simplificar el modelo, es necesario ir actualizando la voxelización. Cada vez que se produce una contracción, se eliminan (dentro de la voxelización) todas las caras que se han movido y se vuelven a introducir las que han sobrevivido.

Es importante señalar que hemos implementado los voxels con un set donde se guardan las caras que contiene. Esto implica que tanto la inserción como el borrado se realizan en tiempo logarítmico en la cantidad de caras almacenadas.

2.3 MOVIMIENTO DE LA NORMAL

Inicialmente, ésta era una comprobación que no teníamos pensado hacer, pero después de las primeras pruebas quedó claro que necesitábamos alguna forma de evitar que las caras girasen demasiado, ya que esto genera geometría que después es difícil de imprimir.

La solución que decidimos usar es bastante simple: Si el realizar una contracción hace que alguna de las normales de las caras que se mueven gire más de α grados, entonces esa contracción es invalida.

El valor de α optimo depende del modelo de entrada, pero hemos visto que en general, valores en el rango de 10° a 35° producen buenos resultados.

Para saber si la normal se ha movido más de lo permitido calculamos el producto escalar de las dos normales (antes y después) y lo comparamos con el valor del coseno de α.

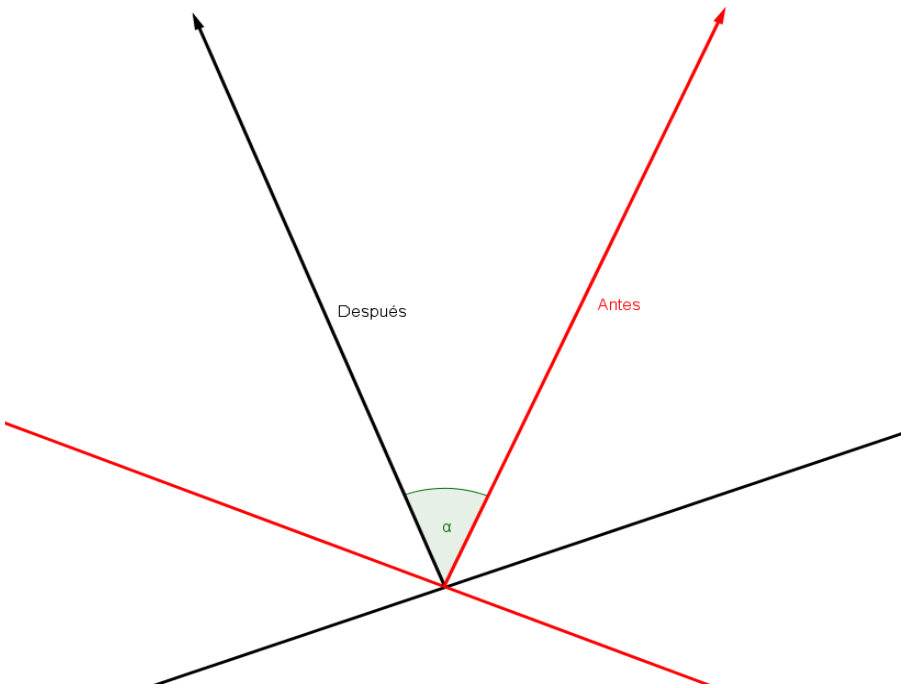

Figura 2.4: Ángulo entre las dos normales. En rojo la cara y normal antes de la contracción, en negro después.

2.4 COSTE ASINTÓTICO

Muchas de las operaciones con mallas de polígonos dependen del grado de conectividad de los vértices, es decir, la cantidad de aristas incidentes en cada vértice. En nuestro caso, teniendo en cuenta que para impresión 3D cuanto menos sea este número mejor y que en mallas de calidad la media será inferior a 6, consideraremos este grado como constante y haremos los cálculos en función del número de vértices N. La cantidad de voxels que pueden interceptar/contener la caja contenedora de una cara asumiremos que es constante ya que, a pesar de depender del tamaño de cada cara, en general son tan pequeñas que no están en más de 4 voxels. Además, también asumiremos que el número de caras es $O(N)$, ya que no hay vértices repetidos.

2.4.1 INICIALIZACIÓN

Solo hay dos cosas que se tendrán que inicializar:

- Quadric metric: La cuádrica (tiempo constante) se tiene que calcular por cada vértice, por lo que tendrá un coste de $O(N)$.

- Voxelización: Por cada cara se tiene que buscar los voxels que la intersectan e insertar si hace falta. Cada comprobación tiene un tiempo constante y, como hemos asumido, hay un número constante de voxels que comprobar. Si se tiene que insertar la cara, esto costará $O(\log N)$; por lo que la operación completa tendrá un coste $O(N \log N)$.

A pesar de que, en el peor de los casos, inicializar puede costar $O(N \log N)$ lo más normal es que en cada voxel haya muy pocas caras y la inserción de una cara se realice en tiempo casi constante.

2.4.2 SIMPLIFICACIÓN

En cada paso de la simplificación se tiene que:

- Comprobar la voxelización $O(1)$ y actualizarla $O(\log N)$.
- Comparar las normales $O(1)$.
- Actualizar las cuádricas modificadas $O(1)$
- Actualizar e insertar en la cola de prioridad $O(\log N)$.

Con esto deducimos que cada paso tiene un coste $O(\log N)$.

Si recordamos el algoritmo, en una iteración puede que no se realice la contracción, en cuyo caso se pasa a la siguiente iteración. En el peor de los casos, no se conseguirá contraer ningún halfedge hasta que a la cola solo le quede un elemento. Después de hacer la contracción se insertará un número constante de elementos (suposición inicial) en la cola y el algoritmo continuará hasta que, en el peor de los casos, se hayan contraído todas las aristas. A partir de esto podemos deducir que el número de iteraciones será lineal con respecto a N, lo que implica que la simplificación tendrá un coste $O(N \log N)$.

3 VISOR 3D

Uno de los objetivos de este proyecto es permitir al usuario apreciar visualmente las diferencias entre el modelo antes y después de la simplificación. Para conseguir esto, hemos desarrollado un visor 3D con varias características que describiremos a continuación.

Después de que se ha seleccionado y cargado un modelo, el visor automáticamente lo presenta al usuario, utilizando iluminación difusa para que se pueda apreciar mejor la profundidad y pintando tanto las caras como las aristas (wireframe), esto último siendo opcional.

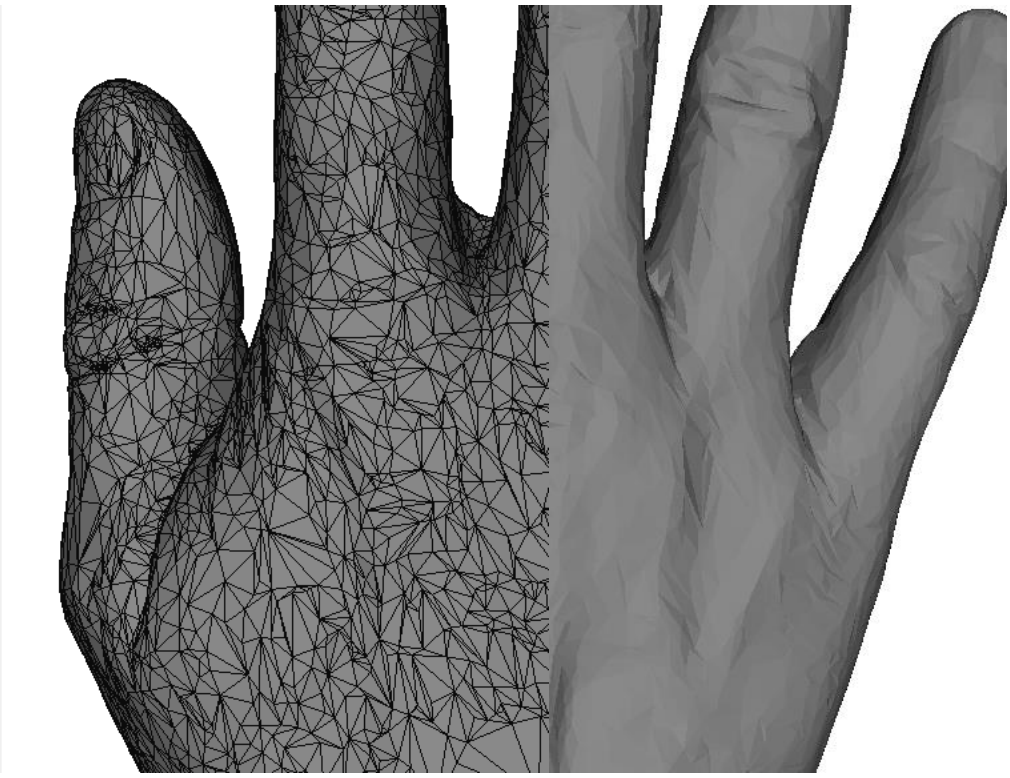

Figura 3.1: Modelo 3D dibujado con wireframe (izquierda) y sin (derecha)

Utilizando la rueda del ratón, se puede acercar y alejar el modelo. Manteniendo presionado el botón izquierdo y moviendo el ratón se puede girar el modelo en los ejes X e Y; de la misma forma, con el botón derecho se puede mover el modelo en las 4 direcciones perpendiculares a la cámara. Con todo esto se permite al usuario visualizar los detalles que le llamen la atención sin tener que utilizar otro programa.

Figura 3.2: Modelo 3D antes de simplificar

Aparte de estas, que son las funcionalidades que se esperan de cualquier visor 3D, también es posible mientras se está realizando la simplificación, ver en tiempo real el estado del modelo; de esta forma se puede apreciar visualmente como trabaja el algoritmo.

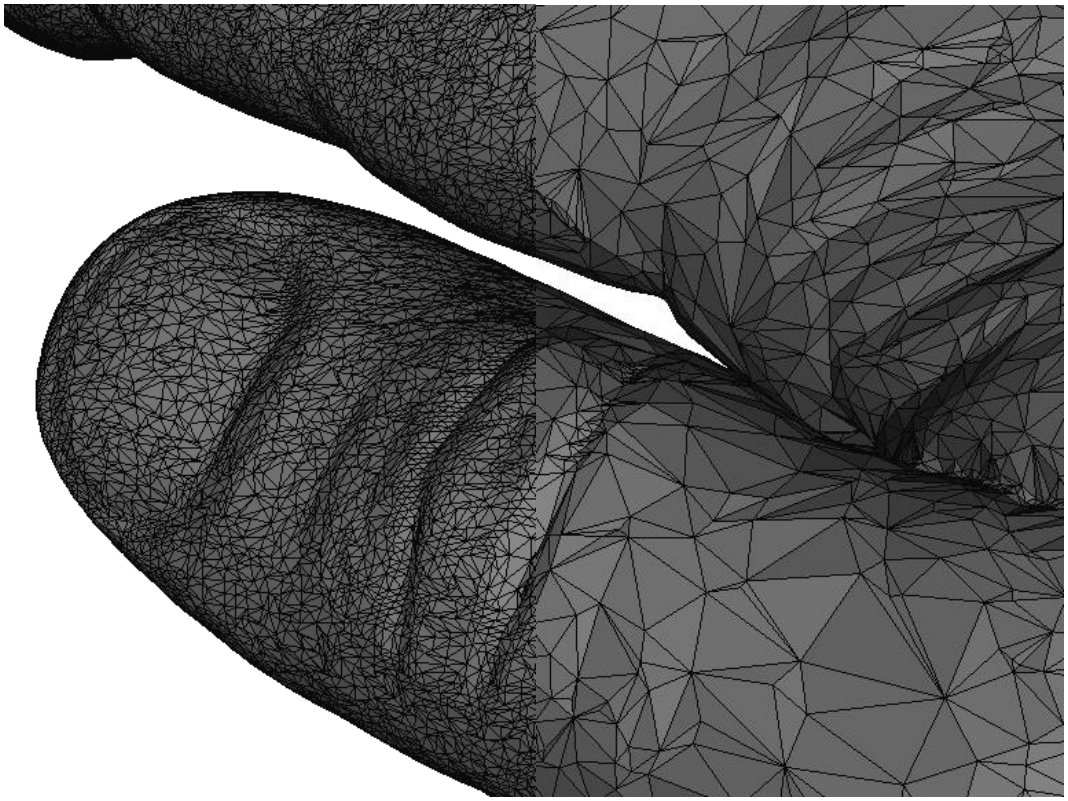

Figura 3.3: Modelo 3D durante (izquierda) y después (derecha) de la simplificación

Por último, una vez realizada la simplificación y con las distancias entre los dos modelos calculadas (en el siguiente apartado se explica el proceso), el visor permite codificarlas como un mapa de calor[3] y pintarlas encima del modelo. El mapa de calor es de 5 colores, donde azul representa distancia 0 y rojo representa la máxima distancia (la diagonal de los voxels).

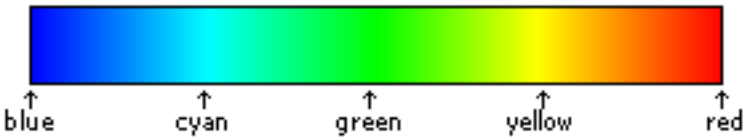

Figura 3.4: Mapa de calor de 5 colores

[3] Un mapa de calor es una representación de datos en forma de mapa o diagrama en el cual el valor de los datos está representado como colores.

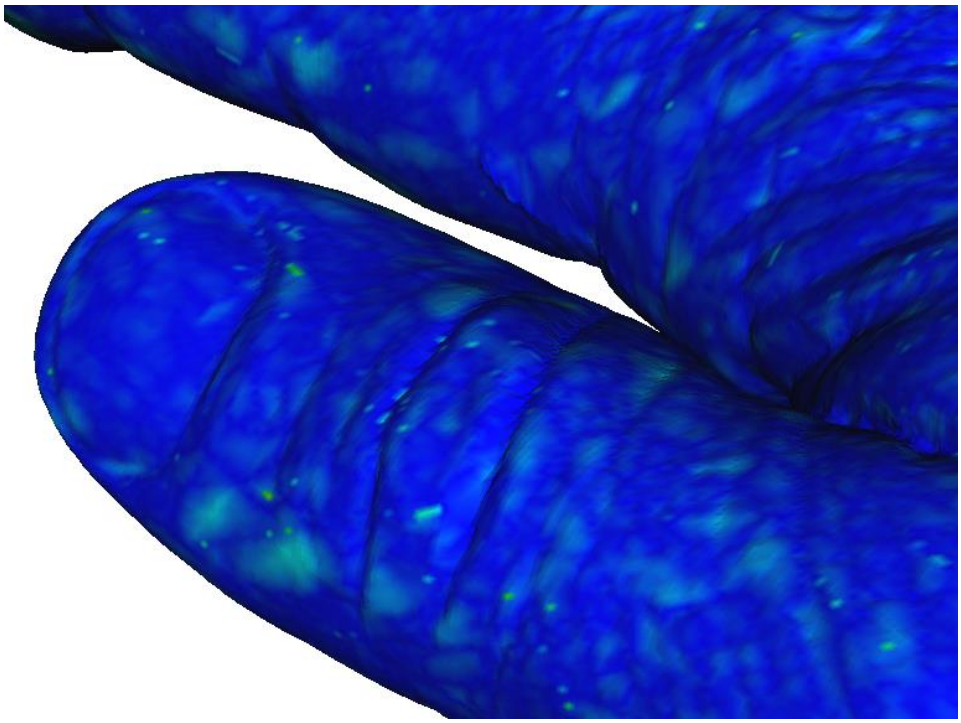

Figura 3.5: Modelo 3D con las distancias codificadas como un mapa de calor

Cabe destacar que la interfaz y la simplificación se procesan en hilos diferentes, por lo que el constante pintado del modelo no afecta a la velocidad con la que se simplifica el modelo, siempre y cuando el procesador tenga más de un núcleo.

3.1 CALCULO DE DISTANCIAS

Una vez terminada la simplificación, se procede a calcular las distancias entre el modelo original y el simplificado. Pseudocódigo del algoritmo:

```
por cada vértice en el modelo original
    se calcula dentro de qué voxel se encuentra
    por cada cara dentro de ese voxel
        calcular la distancia punto triángulo entre el punto y la cara
        guardar la menor de estas distancias
    la menor distancia es la distancia en ese vértice
```

Para el cálculo de la menor distancia punto triángulo utilizamos el código que ya viene con la librería de OpenMesh, pero básicamente el algoritmo consiste en:

```
proyectar el punto P sobre el plano del triángulo y obtener Pt
si Pt está en la región 0
    la distancia es la distancia entre los dos puntos P y Pt
si Pt está en la región 1, 3 o 5
    la distancia es la distancia entre P y el vértice correspondiente
si Pt está en la región 2, 4 o 6
    la distancia es la distancia punto recta entre P y la arista
    correspondiente
```

La distancia entre punto arista se calcula averiguando en que punto de la arista intersecta el vector perpendicular que pasa por P y por la arista.

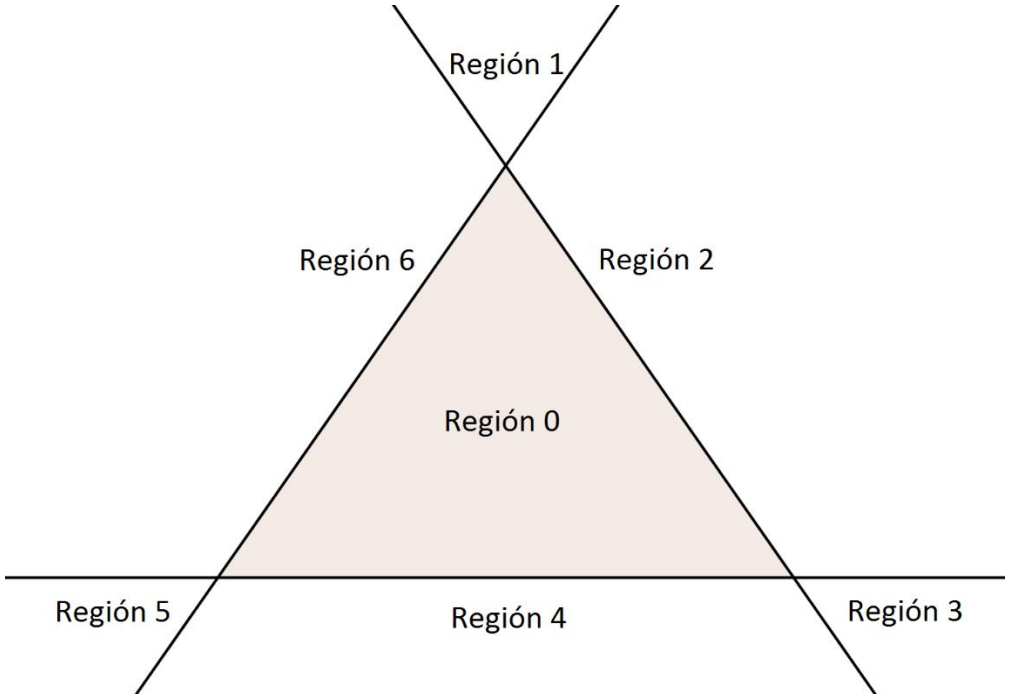

Figura 3.6: Triángulo con las diferentes regiones en las que se puede encontrar la proyección

4 INTERFAZ

Para que el programa sea usable por el usuario final, es necesario que cuente con una interfaz intuitiva y cómoda de utilizar. Esto, aparte de ser uno de nuestros objetivos, es una de las partes más importantes del programa, ya que por muy bien que funcione un programa, si el usuario no sabe usarlo buscará otras alternativas.

4.1 ESPECIFICACIÓN

Para que la interfaz cumpla con los objetivos que nos hemos impuesto, es necesario que posea una serie de funcionalidades:

- Cargar y guardar modelos desde archivos locales.

- Configurar el diámetro del cabezal y la altura de las capas

- Visualizar los modelos cargados (visor 3D)

- Rotar y trasladar los modelos (visor 3D)

- Informar al usuario del estado de la simplificación mediante una barra de progreso

- Que la interfaz se pueda cambiar de tamaño

- No bloquear la interfaz mientras se están simplificando los modelos

- Ver información sobre el resultado de la simplificación

4.2 DISEÑO

Como ya habíamos comentado, hemos usado Qt para programarla. El resultado es simple, pero tiene todas las funcionalidades necesarias. En la siguiente figura se puede observar la interfaz y sus partes:

1- Menú con las opciones de cargar, guardar o salir

2- Configuración de los parámetros de la impresora

3- Opciones para rotar el modelo (no el visor) antes de la simplificación. No está implementado

4- Opciones para escalar el modelo (no el visor) antes de la simplificación. No está implementado

5- Opción para cambiar entre los dos diferentes modos del visor: Modelo con wireframe o modelo original con distancias pintadas

6- Datos sobre la simplificación

7- Botón para ejecutar el algoritmo de simplificación

8- Visor 3D

9- Barra de progreso

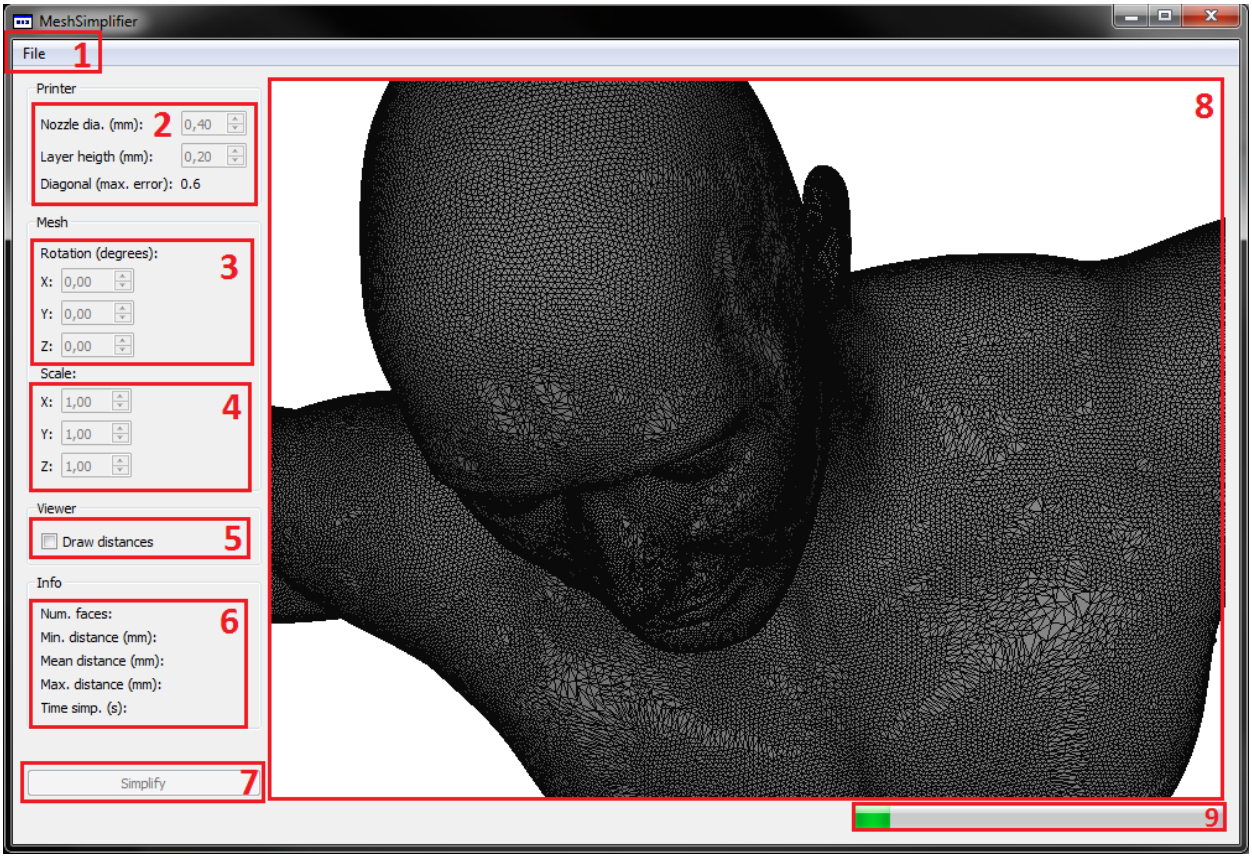

Figura 4.1: Partes de la interfaz

4.3 FUNCIONAMIENTO

Para que la interfaz pueda funcionar es necesario comunicar los diferentes elementos que la componen. En Qt esto se hace mediante Signals y Slots. Un Signal es una especie de mensaje que un componente le puede enviar a otro y estos mensajes son recibidos en los Slots, que son básicamente agujeros de entrada que saben qué hacer cuando reciben un mensaje. Por ejemplo, cuando se presiona el botón de simplificar, este botón le envía un mensaje al visor 3D, que cuando lo recibe sabe que tiene que empezar con la simplificación.

Hemos optado por hacer que el visor 3D sea quien reciba o mande todos los mensajes. Así, el visor es quien se encarga de controlar todos los demás componentes de la interfaz.

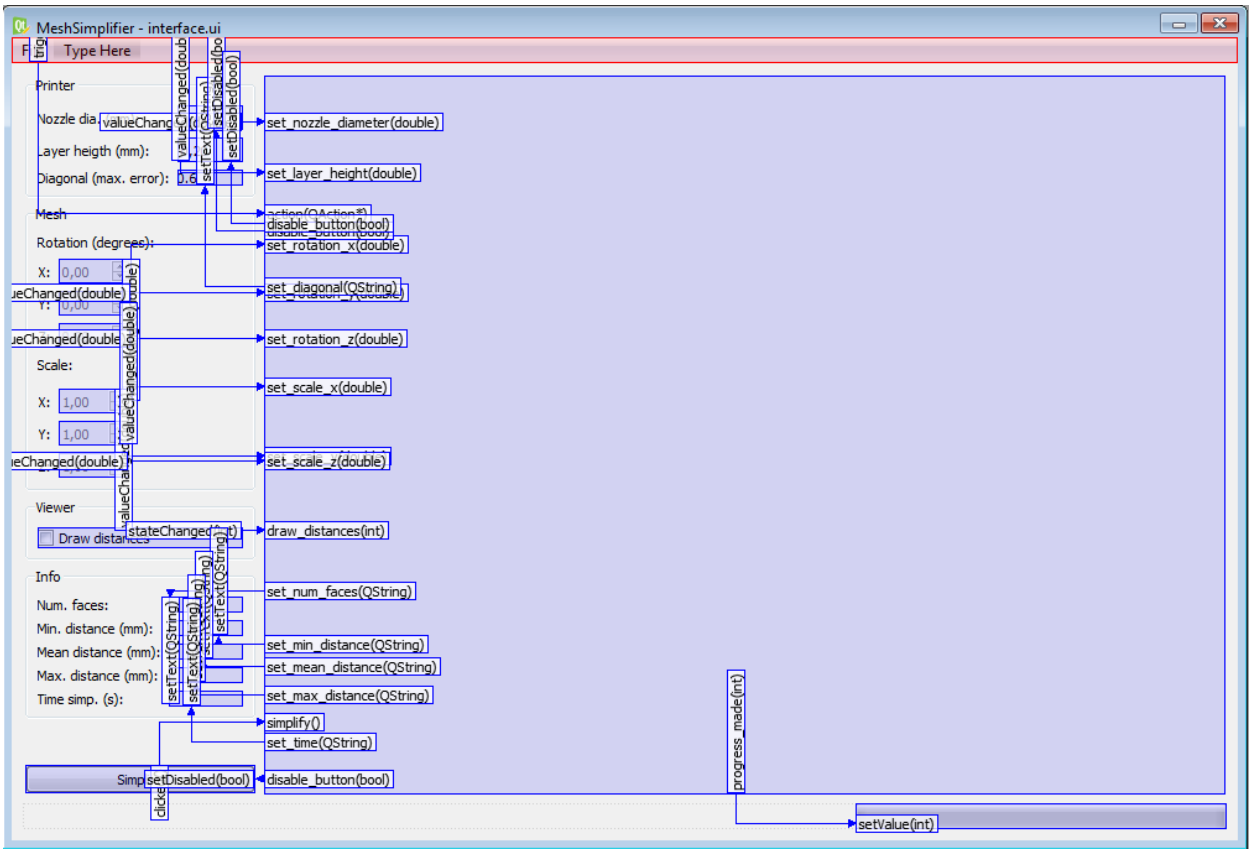

Figura 4.2: Diagrama con los Signals y Slots de la interfaz

5 PRUEBAS Y RESULTADOS

En esta sección comentaremos las pruebas que hemos hecho y los resultados obtenidos con los diferentes modelos y opciones.

Para realizar las pruebas hemos utilizados diez modelos diferentes:

Nombre	Obtenido de	Caras	Tipo
- hand_final.stl	aim@shape	1515706	escaneado
- 70_crank.stl	aim@shape	100048	modelado
- 421_torso_uniform100.stl	aim@shape	284692	modelado
- 803_803_neptune_4Mtriangles_manifold.stl	aim@shape	4007872	escaneado
- 810_Red_circular_box_1.4Mtriangles_clean.stl	aim@shape	1402640	escaneado
- 814_Ramesses_1.5Mtriangles_clean0.2.stl	aim@shape	1652528	escaneado
- Elite Knight - Dark souls -V2.stl	pinshape.com	1465966	escaneado
- knocker_dragon-without_supports_0.5.stl	pinshape.com	1076172	modelado
- lighthouse_01.stl	pinshape.com	398724	modelado
- loubie_aria_dragon.stl	pinshape.com	311898	modelado

Hemos puesto especial atención en que hubiera modelos de diferentes tipos: Escaneados, modelados y una pieza mecánica. Intentando siempre que fuesen modelos con una gran cantidad de caras, de otra forma no tendría sentido simplificarlos.

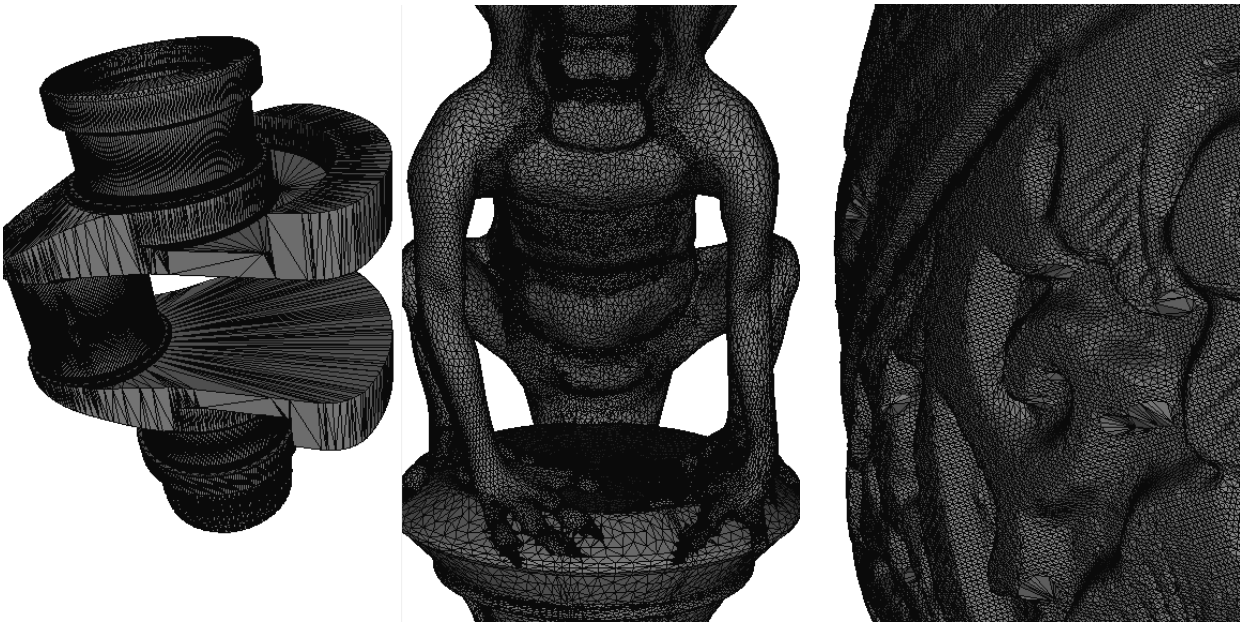

Figura 5.1: Una pieza mecánica (izquierda), modelado con programas CAD (medio) y escaneado (derecha)

Algunos de los modelos escaneados han tenido que ser redimensionados porque eran demasiado grandes para ser impresos y el tamaño en memoria de la voxelización se disparaba:

- 803_803_neptune_4Mtriangles_manifold0.15.stl redimensionado a 0.15
- 814_Ramesses_1.5Mtriangles_clean0.2.stl redimensionado a 0.2
- knocker_dragon-without_supports_0.5.stl redimensionado a 0.5

Como parámetros de la impresora hemos elegido los valores más comunes [16].

Diámetro del cabezal: 0.25mm, 0.3mm, 0.4mm y 0.5mm

Altura de las capas: 0.1mm, 0.2mm y 0.3mm

Todo esto ha resultado en 120 permutaciones diferentes que hemos ejecutado en batch a través de la línea de comandos, guardando los resultados en un log.

5.1 RESULTADOS GENERALES

Empezaremos analizando toda la información que hemos obtenido como un conjunto. Sin entrar en los detalles de cada modelo.

Con cada ejecución hemos obtenido los siguientes datos:

- Número de caras antes y después
- Error medio
- Error máximo
- Tiempo
- Modelo resultante

Usando estos datos hemos generado la tabla 5.1, donde podemos ver que, en general, los tiempos de ejecución son aceptables, el peor tiempo lo tiene neptune, pero es normal al tener más del doble de caras que todos los demás modelos.

El error medio es muy bueno, pero por otro lado en muchas de las ejecuciones el error máximo ha superado la diagonal del voxel, esto no tendría que pasar.

Por último, podemos ver que incluso en el peor de los casos, la cantidad de caras eliminadas es muy significativa, por lo que en este aspecto el algoritmo cumple con su cometido.

Modelo	Tiempo medio simplificación (s)	Error medio (mm)	Media de caras reducidas	Error max. superior a la diagonal
hand_final.stl	95.31	0.0164	98.2%	25%
70_crank.stl	43.65	0.0045	94.5%	0%
421_torso_uniform100.stl	22.61	0.0126	93.1%	0%
803_803_neptune_4Mtriangles_manifold0.15.stl	280.93	0.0119	97.9%	50%
810_Red_circular_box_1.4Mtriangles_clean.stl	87.34	0.0124	74.1%	50%
814_Ramesses_1.5Mtriangles_clean0.2.stl	86.63	0.0088	75.4%	58%
Elite Knight - Dark souls -V2.stl	109.66	0.0088	93.4%	33%
knocker_dragon-without_supports_0.5.stl	76.18	0.0144	81.0%	58%
lighthouse_01.stl	32.65	0.0151	82.5%	8%
loubie_aria_dragon.stl	26.47	0.0147	85.7%	50%

Tabla 5.1: Resumen de las pruebas

5.2 RESULTADOS ESPECÍFICOS

Como ya hemos comentado, nos hemos esforzado en utilizar modelos variados para ver cómo se comporta el algoritmo en diferentes situaciones. Por eso, ahora analizaremos los resultados de algunos casos específicos que nos han llamado la atención.

5.2.1 HAND_FINAL

Este es un modelo escaneado de una mano, por lo tanto, orgánico. Su superficie es bastante plana y en general las curvas son suaves; tiene espacios donde hay ángulos entre dos caras muy cerrados (espacio entre dos dedos).

Podemos ver que en general el algoritmo se ha comportado correctamente, simplificando más las partes planas y menos las que tienen ángulos cerrados. En la base, con un ángulo de 90 grados es donde se pueden ver las zonas con mayor error.

Figura 5.2: hand_final con el mapa de calor

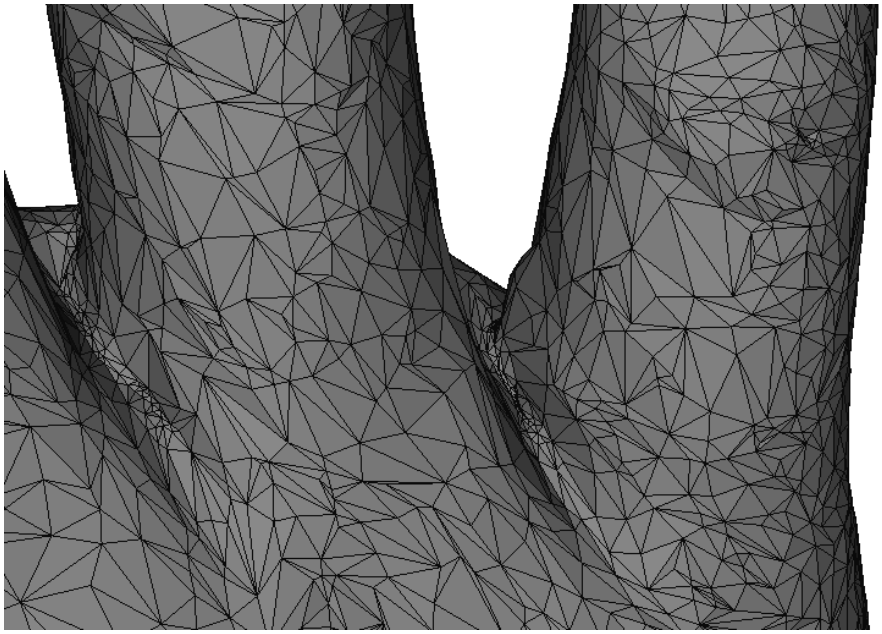

Figura 5.3: Partes de hand_final donde el algoritmo ha evitado simplificar

5.2.2 RED_CIRCULAR_BOX

Este es un modelo escaneado de un objeto esferico con grabados. Tiene muchas curvas, detalles pequeños y ángulos cerrados, pero también hay partes planas.

Este es uno de los modelos que menos se ha conseguido simplificar, esto se debe a todos los detalles que tiene y que el algoritmo ha decidido preservar. Hay algunos que son demasiado pequeños para que se puedan apreciar al imprimirse y se han mantenido, seguramente, debido al cambio que produciría en las normales.

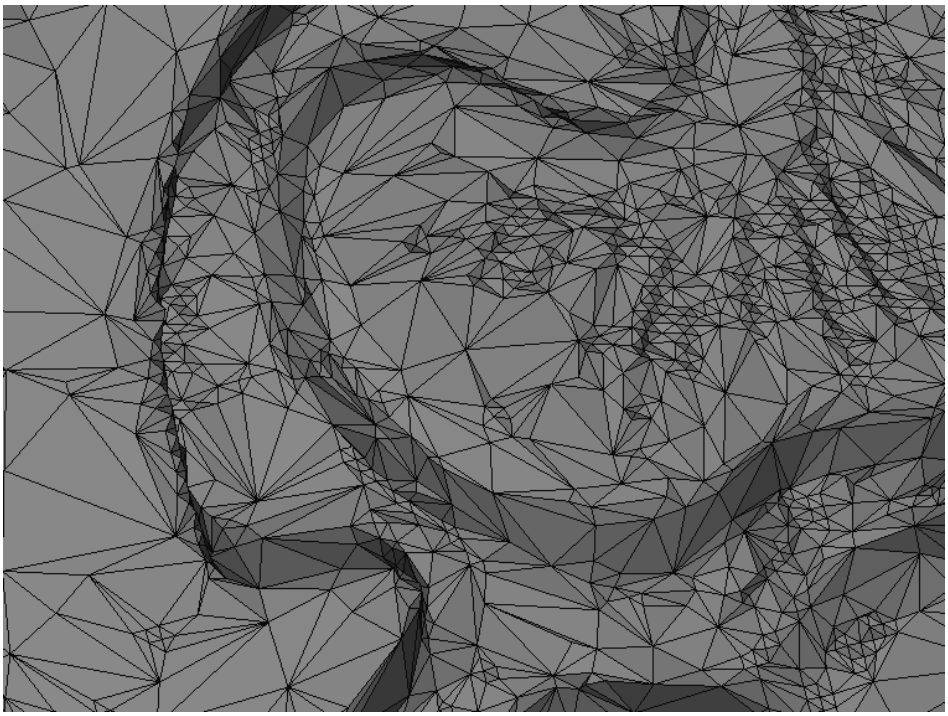

Figura 5.4: Detalles en Red_circular_box

5.2.3 CRANK

Este modelo es una pieza mecánica, con muchas superficies circulares, ángulos de 90 grados, triángulos con aristas muy largas y vértices con muchas aristas incidentes.

Es una malla bastante complicada y no sorprende que el algoritmo no haga un muy buen trabajo, hay partes donde la deformación es considerable. Un ejemplo de esto es la figura 5.5, donde se puede observar que el modelo simplificado se aleja bastante del original y, además, el mapa de calor (figura 5.6) no lo detecta.

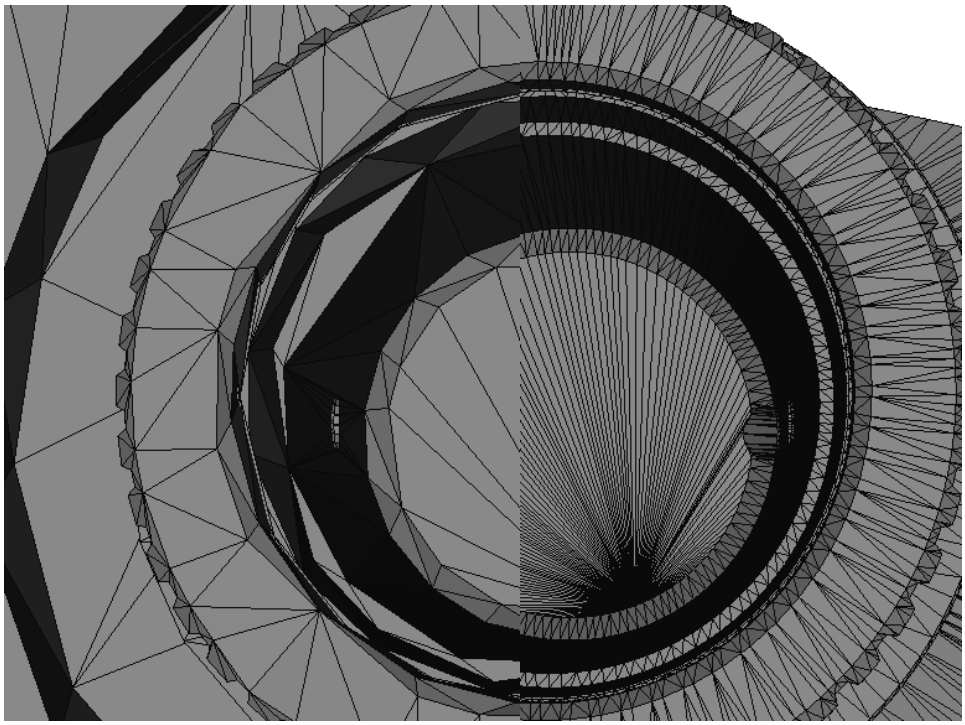

Figura 5.5: Crank antes (derecha) y después (izquierda) de simplificar

Figura 5.6: Crank con el mapa de calor pintado

Aun así, no todo es malo, los ángulos de 90 grados se han conservado bastante bien.

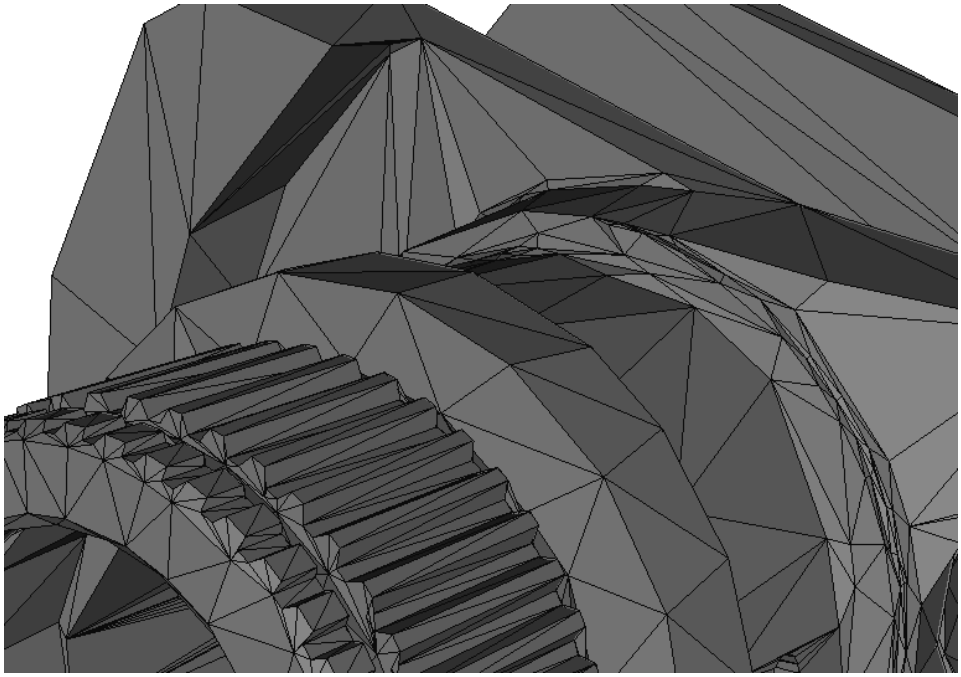

Figura 5.7: Exterior de Crank simplificado

6 CONCLUSIONES FINALES

Habiendo podido experimentar con el programa y analizado los resultados del algoritmo podemos sacar varias conclusiones, tanto de las cosas que han funcionado como las que se podrían mejorar.

Primero, y esto era uno de los objetivos, el algoritmo permite reducir drásticamente la complejidad de todos los tipos de modelos que hemos probado. Tan es así que seguramente podríamos poner condiciones más duras a la hora de contraer aristas e intentar, de esta forma, paliar algunos de los problemas que comentaremos más adelante.

En lo referente a la interfaz, se han cumplido todos los objetivos. Es intuitiva, fácil de usar e incluye todas las funcionalidades deseadas. El poder ver cómo se va simplificando la malla en tiempo real es interesante.

Otro de los puntos fuertes de nuestro algoritmo es lo bien que preserva los bordes, dejando más geometría en estas áreas para evitar que se deforme la silueta del modelo.

El tiempo de ejecución es aceptable. Se tendría que comparar con algoritmos similares para ofrecer una valoración más precisa en este aspecto.

Por otro lado, aunque el error medio de todos los puntos es bastante pequeño, no hemos conseguido que el error máximo sea inferior a la diagonal de los voxels. Esto se debe a que los voxels en realidad son más grandes, ya que están superpuestos.

También hemos podido ver que con solo medir la distancia entre las dos mallas desde un lado no es suficiente, existen casos en que se está subestimando el error.

En resumen, el resultado del proyecto es satisfactorio ya que se han alcanzado todos los objetivos. Si bien hemos podido comprobar que en algunos casos el algoritmo no funciona todo lo bien que se desearía, en general, da buenos resultados.

7 TRABAJO FUTURO

A pesar de haber alcanzado los objetivos, hay muchas formas en las que se podría mejorar el programa, ya sea mejorando funcionalidades o agregando de nuevas. A continuación, explicaremos algunas de estas ideas:

- **Uso de la interfaz más intuitivo, trackball:** Trackball es un método que permite rotar objetos en un visor 3D. A diferencia de la implementación actual, permite rotar en los 3 ejes y, además, es más fácil de usar

- **Agregar las opciones de escalado y rotación del modelo antes de procesado:** Estas opciones ya están visibles en la interfaz, pero no han sido implementadas por falta de tiempo. Terminarlas evitaría que el usuario tenga que cambiar de programa cada vez que desea usarlas.

- **Comprobaciones para evitar generar geometría non-manifold:** Geometria non-manifold es geometría que no puede existir en el mundo real, por ejemplo: zonas sin grosor, caras internas, vértices desconectados. Este tipo de geometría provoca problemas al imprimir los modelos, por lo que es interesante que no se generé. Asegurar que esto se cumpla es bastante complicado y hacen falta una serie de comprobaciones locales y globales.

- **Agregar el histograma de las distancias a la interfaz:** Poseer un histograma del mapa de calor en la interfaz ayudaría a identificar, de forma más rápida, la calidad del modelo simplificado.

- **Calculo del error desde los dos lados:** Actualmente solo se calcula el error desde la malla original a la simplificada y, como hemos visto, esto no siempre produce el resultado deseado. Si se calcula desde los dos lados el resultado es más preciso. Para implementarlo será necesario subdividir la malla antes de buscar la cara más cercana en cada vértice.

- **Voxelización sin voxels superpuestos:** Para evitar ciertas situaciones degeneradas, durante la implementación optamos por hacer que una parte de cada voxel este superpuesta a sus vecinos. Esto ha causado que el error máximo sea mayor que el deseado, por lo que se tendrían que buscar alternativas que funcionen para los dos problemas.

8 GESTIÓN DEL PROYECTO

8.1 ALCANCE Y METODOLOGÍA

En esta sección definiremos las funcionalidades que deberá poseer la aplicación final para cumplir todos los objetivos que nos hemos propuesto, los obstáculos que nos podemos encontrar al tratar de implementarlas y hablaremos sobre las metodologías que usaremos tanto para trabajar como para asegurarnos de que nuestros resultados son correctos.

8.1.1 ALCANCE

Para resolver nuestro problema necesitamos una aplicación que sea capaz de cargar un modelo 3D en los diferentes formatos usados actualmente por los slicers, stl es el estándar. Una vez cargada la malla se tiene que poder introducir los parámetros de nuestra impresora 3D: diámetro del cabezal y espesor de las capas. Opcionalmente también podemos incluir más preferencias, como podría ser la desviación máxima de las normales.

Una vez introducidos todos los datos, la aplicación debe dar la opción al usuario de empezar la simplificación. Mientras la simplificación se lleva a cabo, la interfaz debe tener una barra de progreso que permita saber al usuario que el programa sigue funcionando.

Cuando la simplificación haya terminado, se debe avisar al usuario y enseñar el resultado en el visor 3D. El visor 3D debe permitir mover y girar el modelo para observarlo desde diferentes ángulos y la superficie del modelo debe pintarse codificando su distancia al modelo original. Si el usuario no está satisfecho con el resultado, debe poder cambiar los parámetros (o el modelo) y volver a empezar todo el proceso.

Si el usuario así lo desea debe disponer de la opción de guardar el resultado en, por lo menos, formato stl.

8.1.2 POSIBLES OBSTÁCULOS

En esta sección describiremos los principales obstáculos que podemos encontrarnos durante el desarrollo de esta aplicación y las posibles soluciones.

Errores en el código: Al programar, por mucho cuidado que se ponga, se introducen errores, esto es algo que no se puede evitar. Para corregir estos errores realizaremos pruebas cada vez que algún módulo del código se termine.

Que el cálculo de las distancias sea muy lento: Según lo rápido que vaya el algoritmo y, sobretodo, si el usuario elige una subdivisión muy grande, la tarea de calcular la distancia de todos los puntos puede ser muy lenta. Si consideramos que esto es un problema podemos paralelizar esta sección del código para aliviarlo.

Que el modelo simplificado no sea imprimible: Asumiendo que el modelo de entrada sea correcto y el software de la impresora lo acepte, existe la posibilidad de que después de simplificarlo, se generen problemas en la superficie que nos obliguen a repáralo antes de poder usarlo. Los principales problemas son: auto intersección entre caras y agujeros.

8.1.3 METODOLOGÍA Y RIGOR

Para el desarrollo de este proyecto hemos dividido el trabajo en tareas realizado un plan aproximado que nos facilitará la organización del trabajo. En esta sección describiremos este plan, las herramientas de seguimiento y los métodos de validación que utilizaremos.

8.1.3.1 Metodología de trabajo

Lo primero que haremos será diseñar una primera versión de la interfaz que contenga todos los botones y opciones que pensamos incluir y con el visor 3D. En esta primera fase la única funcionalidad de la aplicación será cargar un modelo, dibujarlo y permitir moverlo. Temporalmente, agregaremos la opción de permitir cargar dos modelos para poder compararlos.

Una vez la interfaz esté terminada pasaremos a agregarle la funcionalidad de comparar dos modelos al visor. Tener el visor funcionando antes que el algoritmo de simplificación es necesario ya que sin él nos será difícil observar los resultados. Más adelante ya especificaremos los detalles, pero al terminar la programación de esta etapa haremos pruebas para comprobar que la comparación da resultados correctos.

Con el visor ya funcionando pasaremos a programar el algoritmo de simplificación, pasar los resultados al visor y guardarlos en un archivo. Esta etapa será la que más tiempo y esfuerzo requerirá. Al final también será necesario hacer pruebas para asegurarnos de que funciona correctamente.

Con prácticamente todas las funcionalidades implementadas en la última etapa aprovecharemos la experiencia que tendremos con la aplicación para terminar la interfaz y asegurarnos de que sea intuitiva y fácil de usar. Si vemos que hay alguna característica que puede mejorar la aplicación podemos incluirla.

8.1.3.2 Herramientas de seguimiento

A pesar de que solo habrá una persona trabajando con el código, utilizaremos Github para el control de versiones ya que previene la perdida de código y ayuda con la corrección de errores.

Para ayudar con la organización de las diferentes tareas y asegurarnos de seguir el horario propuesto usaremos un diagrama de Gantt.

Para la comunicación con el director del proyecto usaremos correo electrónico.

8.1.3.3 Métodos de validación

Hay dos partes del proyecto que necesitan validación: el visor 3D y el algoritmo de simplificación.

Para validar los resultados del visor 3D, haremos pruebas pasándole los dos modelos directamente. Uno de ellos modificado ligeramente con una herramienta CAD (Blender), de forma que sepamos cual es el resultado que tiene que dar. Este proceso lo realizaremos varias veces con modelos y modificaciones diferentes, hasta que estemos convencidos de que funciona correctamente.

Validar el resultado de la simplificación es un poco más complicado. Por un lado, para evaluar las diferencias con el original contamos con el visor 3D, donde podremos observar los resultados de diferentes modelos y parámetros. Además, para comprobar que la reducción no empeora la calidad después de la impresión será necesario imprimir varios modelos y comparar los resultados.

8.2 PLANIFICACIÓN TEMPORAL

A continuación, hablaremos sobre la planificación inicial del proyecto, los cambios que se han producido y la planificación final. Justificaremos los cambios y como afectan estos a los objetivos del proyecto.

8.2.1 DESCRIPCIÓN DE LAS TAREAS

En esta sección hablaremos sobre cada una de las tareas en las que hemos dividido el proyecto.

- Hito inicial

Su objetivo es ayudarnos con la planificación y documentación del proyecto. Está compuesta por los siguientes apartados:

1. Definición del alcance y contextualización
2. Planificación temporal

3. Gestión económica y sostenibilidad
4. Presentación preliminar
5. Pliego de condiciones
6. Documentación presentación final

- Configuración del entorno

Antes de poder empezar a trabajar es necesario instalar todos los programas que vamos a utilizar y configurarlos correctamente. Haciendo esto al principio nos aseguramos de que cuando empecemos a trabajar podremos concentrarnos en las tareas correspondientes.

Será necesario instalar MinGW, Qt, Blender, Git, crear un proyecto en GitHub y descargar el código fuente de OpenMesh.

- Primera versión de la interfaz

Usando Qt crearemos una primera versión de la interfaz que contenga todos los elementos necesarios, aunque a lo mejor no en el lugar correcto. Lo más importante de esta fase es hacer que el programa permita cargar modelos, dibujarlos en el visor y que se puedan mover y rotar.

Como ya lo hemos hecho en clase, la implementación de las funciones base del visor no deberían costar mucho.

- Visor 3D

Una vez la interfaz esté terminada pasaremos a agregarle la funcionalidad de comparar dos modelos al visor. Dividiremos esta tarea en tres partes:

Análisis: Pensar cual es la mejor forma de implementar el algoritmo de comparación.

Programación: Como el nombre indica, esta es la fase principal de esta tarea y es donde invertiremos más tiempo.

Pruebas: Una vez terminada la implementación tenemos que probar el resultado. Es posible que si encontramos algún error tengamos que volver a programar, pero tendrían que ser simples retoques. Para las pruebas usaremos Blender y seguramente algún modelo descargado de internet.

- Algoritmo de simplificación

Con el visor ya terminado podemos pasar a programar el algoritmo de simplificación, que es la base de este proyecto. Esta tarea, como la anterior, se divide en tres fases:

Análisis: Pensar en la mejor forma de implementar el algoritmo y familiarizarse con la librería.

Programación: Ponerse a implementar el algoritmo y las estructuras de datos necesarias. Probablemente esta sea la tarea que más tiempo consumirá de todo el proyecto.

Pruebas: Con todas las funcionalidades importantes de la aplicación funcionando, hay que comprobar que no hay errores y probar si el algoritmo funciona como esperado. Usaremos el visor para comparar modelos y una impresora 3D para probar los resultados.

- Funcionalidades opcionales

Esta tarea la hemos reservado para que, si en el transcurso del proyecto, hemos visto que es necesario implementar alguna restricción más o agregar algún parámetro podamos hacerlo.

- Versión final de la interfaz

Con el programa ya prácticamente terminado, y después de haber podido probarlo durante un buen tiempo, intentaremos retocar la interfaz para que sea bonita, intuitiva y fácil de usar.

- Hito final

Hemos reservado este tiempo para terminar toda la documentación, preparar la presentación y atar todos los cabos sueltos.

Seguramente haya más tiempo del que hemos usado para terminar el proyecto, pero hemos preferido dejar días libres al final para tener un margen con el que resolver posibles inconvenientes.

8.2.2 Diagrama de Gantt inicial

	Nombre	Duración	Inicio	Fin
1	**Hito inicial**	30d	**22/02/2016**	**01/04/2016**
2	Definición del alcance y contextualización	8d	22/02/2016	02/03/2016
3	Planificación temporal	4d	03/03/2016	08/03/2016
4	Gestión económica y sostenibilidad	5d	09/03/2016	15/03/2016
5	Presentación preliminar	4d	16/03/2016	21/03/2016
6	Pliego de condiciones	7d	24/03/2016	01/04/2016
7	Documentación presentación final	7d	24/03/2016	01/04/2016
8	Configuración del entorno	2d	04/04/2016	05/04/2016
9	Primera versión de la interfaz	3d	06/04/2016	08/04/2016
10	**Visor 3D**	17d	**11/04/2016**	**03/05/2016**
11	Análisis	3d	11/04/2016	13/04/2016
12	Programación	10d	14/04/2016	27/04/2016
13	Pruebas	4d	28/04/2016	03/05/2016
14	**Algoritmo de simplificación**	20d	**04/05/2016**	**31/05/2016**
15	Análisis	2d	04/05/2016	05/05/2016
16	Programación	12d	06/05/2016	23/05/2016
17	Pruebas	6d	24/05/2016	31/05/2016
18	Funcionalidades adicionales	3d	01/06/2016	03/06/2016
19	Versión final de la interfaz	2d	06/06/2016	07/06/2016
20	Hito final	7d	08/06/2016	16/06/2016

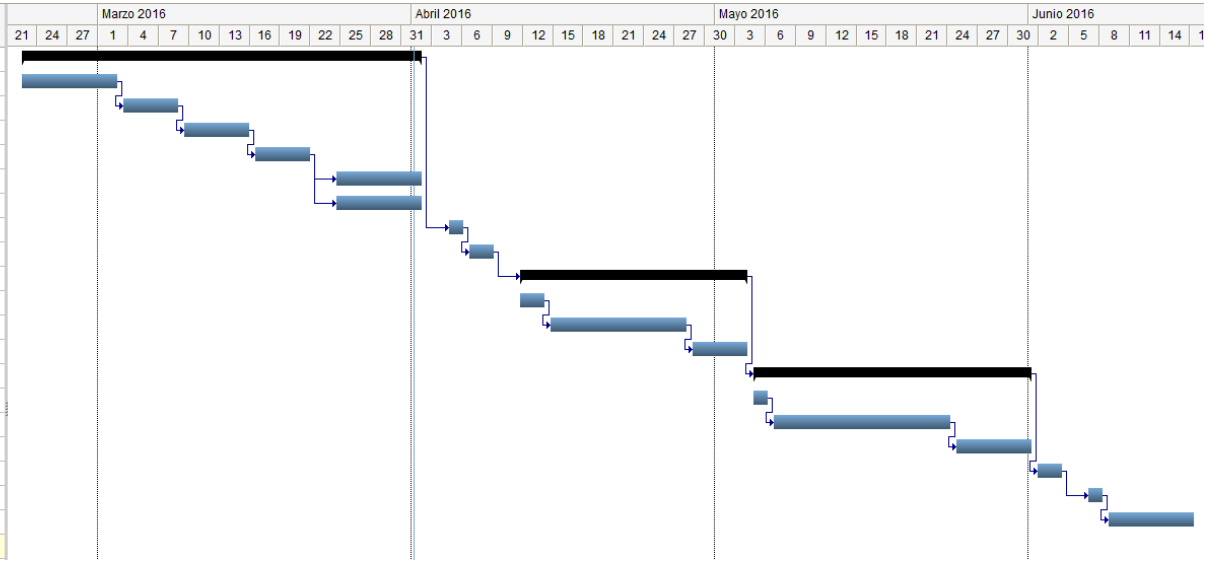

Figura 8.1: Diagrama de Gantt inicial del proyecto

8.2.3 Desviaciones

Durante la ejecución del proyecto, ha habido una serie de cambios o desviaciones en la planificación inicial:

Visor 3D: Un tiempo después de iniciar esta tarea, decidimos posponerla y trabajar primero en el algoritmo de simplificación. El motivo es que el algoritmo de simplificación es la tarea que más tiempo requiere y como las pruebas con impresoras 3D pueden tardar más de lo esperado es mejor hacerlas lo antes posible. Además, después de haber podido analizar el trabajo necesario hemos decidido reducir la cantidad de horas que creemos que serán necesarias para terminarla.

Funcionalidades opcionales: Inicialmente era una tarea opcional, pero después de ver cómo avanza el proyecto hemos decidido eliminarla ya que no tendremos suficiente tiempo para llevarla a cabo.

Aparte de los cambios comentados, también hemos decidido posponer la presentación de la memoria hasta el turno de octubre. El motivo es que consideramos que no teníamos suficiente tiempo con los exámenes finales y trabajo de media jornada para terminar el proyecto a tiempo.

8.2.4 Planificación final

Teniendo en cuenta las desviaciones que han ocurrido durante el desarrollo del proyecto hemos elaborado una planificación final en la que se ven reflejadas (Figura 2):

- Posponer la implementación del visor 3D y acortar el tiempo necesario.

- Quitar la tarea opcional “Funcionalidades adicionales” del calendario.

- Posponer el proyecto durante el tiempo que ha estado parado.

Como podemos observar ninguna de las modificaciones afecta los objetivos ni alcance del proyecto, por lo que siguen siendo los mismo que expusimos en el hito inicial. Sin embargo, al reducir las horas han cambiado los costos en recursos humanos; estos cambios se verán reflejados en el apartado de gestión económica.

	Nombre	Duración	Inicio	Fin
1	**Hito inicial**	30d	**22/02/2016**	**01/04/2016**
2	Definición del alcance y contextualización	8d	22/02/2016	02/03/2016
3	Planificación temporal	4d	03/03/2016	08/03/2016
4	Gestión económica y sostenibilidad	5d	09/03/2016	15/03/2016
5	Presentación preliminar	4d	16/03/2016	21/03/2016
6	Pliego de condiciones	7d	24/03/2016	01/04/2016
7	Documentación presentación final	7d	24/03/2016	01/04/2016
8	Configuración del entorno	2d	04/04/2016	05/04/2016
9	Primera versión de la interfaz	3d	06/04/2016	08/04/2016
10	**Algoritmo de simplificación**	377d	**11/04/2016**	**19/09/2017**
11	Análisis	2d	11/04/2016	12/04/2016
12	Programación	12d	31/08/2017	15/09/2017
13	Pruebas	2d	18/09/2017	19/09/2017
14	**Visor 3D**	10d	**20/09/2017**	**03/10/2017**
15	Análisis	2d	20/09/2017	21/09/2017
16	Programación	6d	22/09/2017	29/09/2017
17	Pruebas	2d	02/10/2017	03/10/2017
18	Versión final de la interfaz	2d	04/10/2017	05/10/2017
19	Hito final	7d	06/10/2017	16/10/2017

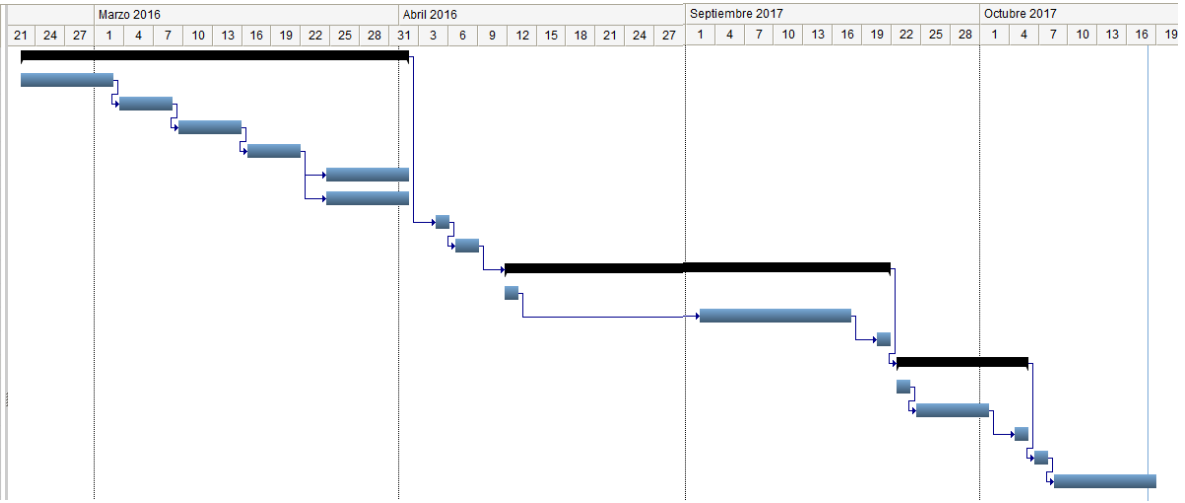

Figura 8.2: Diagrama de Gantt final del proyecto

8.3 GESTION ECONÓMICA

En esta sección detallaremos el presupuesto del proyecto, lo dividiremos en sus partes y explicaremos los conceptos que las componen. Además, hablaremos sobre las medidas que podemos tomar para controlarlo y que no exceda las estimaciones.

8.3.1 PRESUPUESTO DE RECURSOS HUMANOS

Este proyecto será desarrollado por una sola persona, sin embargo, tendrá que desempeñar diferentes roles. En la siguiente tabla podemos ver el tiempo que será necesario en cada uno de los roles, el precio por hora y los totales.

Rol	Horas	Precio por hora	Precio
Director de proyecto	150 h	50 €/h	7500 €
Diseñador	60 h	35 €/h	2100 €
Programador	130 h	35 €/h	4550 €
Tester	20 h	30 €/h	600 €
Total	360 h		14750 €

Tabla 2: Presupuesto de recursos humanos

8.3.2 PRESUPUESTO DE HARDWARE

En la siguiente tabla describiremos el presupuesto dedicado a hardware.

Producto	Precio	Vida útil	Amortización
PC	1500 €	4 años	125 €
Total			125 €

Tabla 3: Presupuesto de hardware

8.3.3 Presupuesto de software

En la siguiente tabla describiremos el presupuesto dedicado a software.

Producto	Precio	Vida útil	Amortización
Office Word	270 €	3 años	30 €
LATEX	0 €	3 años	0 €
Windows 7	130 €	3 años	14 €
Visual Studio 2015	0 €	3 años	0 €
OpenMesh	0 €	3 años	0 €
Qt	0 €	3 años	0 €
Blender	0 €	3 años	0 €
Git	0 €	3 años	0 €
Total			44 €

Tabla 4: Presupuesto de software

8.3.4 Presupuesto de servicios

Para realizar este proyecto necesitaremos utilizar un servicio de impresión para poder probar los resultados. En la siguiente tabla podemos ver el presupuesto que se le dedica.

Producto	Cantidad	Precio por servicio	Precio
Impresión 3D	3	20 €	60 €
Total			60 €

Tabla 5: Presupuesto de servicios

8.3.5 Presupuesto gastos indirectos

En la siguiente tabla citaremos los gastos indirectos en los que incurrimos.

Producto	Cantidad	Precio	Precio
Electricidad	1.8 kWh x 360 h	0.15 €/kWh	97 €
Internet	4 meses	30 €/mes	120 €
Local + Agua	4 meses	100 €/mes	400 €
Total			617 €

Tabla 6: Presupuesto de servicios

8.3.6 PRESUPUESTO TOTAL

Si sumamos todos los gastos obtenemos el presupuesto total, que podemos ver en la siguiente tabla.

Concepto	Precio
Recursos humanos	14750 €
Hardware	125 €
Software	44 €
Servicios	60 €
Gastos indirectos	617 €
Total	15596 €

Tabla 7: Presupuesto total

8.3.7 CONTROL DE DESVIACIONES

Si ocurre alguna desviación en el presupuesto es seguro que el causante será un retraso en alguna de las tareas que componen el proyecto. Para intentar solucionar esto, y que los gastos no superen al presupuesto, contamos con la opción de disminuir el tiempo asignado a otras tareas, o incluso podemos suprimir la inclusión de funcionalidades opcionales y quedarnos con la primera versión de la interfaz.

8.4 SOSTENIBILIDAD Y COMPROMISO SOCIAL

En esta sección vamos a evaluar la sostenibilidad del proyecto en cada una de sus tres diferentes facetas.

8.4.1 SOSTENIBILIDAD AMBIENTAL

Durante la etapa de desarrollo, este proyecto hará uso de un ordenador y, calculamos, unos 1.8 kWh durante 360 horas.

Al ser un producto de software, una vez terminado el desarrollo ya no tiene un impacto directo sobre el medio ambiente, sin embargo, sí que posee un impacto indirecto. Tanto en la fase de desarrollo como durante su vida útil esta aplicación consume electricidad, la cual es generada, en parte, con combustibles fósiles. Además, desarrollar aplicaciones para impresoras 3D promueve esta práctica, mediante la cual se genera basura y se gasta más electricidad.

También podríamos decir que esta aplicación ayuda a evitar impresiones fallidas, cuyo producto irá directamente a la basura. En este aspecto puede ser algo positivo para el medio ambiente.

El uso de materiales reciclables durante la impresión es una práctica que se empieza a ver, aunque no es lo más usual. En cualquier caso, esta aplicación no tiene influencia sobre los materiales usados.

8.4.2 Sostenibilidad económica

Desarrollar este proyecto tiene un coste aproximado de 15596 €, esto hace que sea económicamente no sostenible, ya que la única manera de hacer que la gente use la aplicación es ofrecerla gratuitamente. En caso de cobrar por su uso, aparte de que casi nadie pagaría, deberíamos pagar por las licencias de varias librerías usadas, con lo que aumentaría aún más el precio del desarrollo.

El motivo por el que considero que nadie pagaría por este producto es porque hay productos similares (simplificadores) disponibles gratuitamente y aunque a lo mejor no producen un resultado tan bueno, al público al que va dirigido la aplicación (aficionados) no le importará.

Como ya hemos explicado en el apartado de planificación, hemos necesitado menos tiempo del inicialmente planificado, lo que ha resultado en el abaratamiento de los costes de producción.

8.4.3 Sostenibilidad social

Es sector de las impresiones 3D está en constante desarrollo y una aplicación que lo facilite solo puede promoverlo. Sobre todo si es gratuita, nuestra aplicación puede facilitar la vida de los posibles usuarios, haciendo que pierdan menos tiempo y dinero.

Todas las soluciones no profesionales que hacen algo parecido son gratuitas, por lo que no hay ningún usuario ni colectivo que pueda salir perjudicado

8.4.4 Matriz de sostenibilidad

A continuación, presentamos la matriz de sostenibilidad, a la cual le hemos asignado valores según nuestra opinión del proyecto.

¿Sostenible?	Económica	Social	Ambiental
Planificación	Viabilidad económica	Mejora calidad de vida	Viabilidad ambiental
Valoración (0:10)	2	6	9
Resultados	Coste final versus previsión	Impacto social	Consumo de recursos
Valoración (-10:10)	1	6	9
Riesgos	Adaptación a cambios	Daños sociales	Daños ambientales
Valoración (-20:0)	0	0	0
Valoración total	33		

Tabla 8: Matriz de sostenibilidad

8.5 IDENTIFICACIÓN DE LEYES Y REGULACIONES

La única ley que afecta a este proyecto es la Ley de Propiedad Intelectual. Por un lado, establece que tanto el código fuente, binarios y documentación del proyecto están protegidos por derechos de autor, en este caso: alumno, director del proyecto y la universidad; pero más importante es tener en cuenta que varios elementos usados en el proyecto también poseen estos derechos: Qt (GPL) y OpenMesh (BDS). Afortunadamente, sus licencias nos permiten usar, modificar, distribuirlos y comercializar nuestro producto siempre y cuando incluyamos su licencia al distribuir el código o archivos binarios (Qt no requiere ni eso).

En lo referente a visualizar o modificar un modelo 3D no hay ninguna ley que lo regule aparte de la ya citada. Esto implica que el usuario podrá usar el programa con cualquier modelo 3D siempre y cuando su licencia (del modelo) lo permita.

Todos los modelos que hemos utilizado para hacer las pruebas son distribuidos gratuitamente con licencia no comercial, por lo que no supone ningún problema.

9 REFERENCIAS

[1] 3ders.org. (2016). Price compare - 3D printers. Obtenido el 30 de marzo del 2016, de http://www.3ders.org/pricecompare/3dprinters/

[2] 3dprintingforbeginners.com. (2014). 3D Printing Technology Explained. Obtenido el 30 de marzo del 2016, de http://3dprintingforbeginners.com/3d-printing-technology/#more-1782

[3] Hamel, M. (2015). Improving print quality by reducing triangle count. Obtenido el 30 de marzo del 2016, de https://re3d.org/reduce-stl-triangles-using-autodesk-meshmixer/

[4] Knapp, M. (2002). Mesh Decimation using VTK. *Network*, *vi*, 1–8. Obtenido de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.9441&rep=rep1&type=pdf

[5] Garland, M., & Heckbert, P. S. (1997). Surface simplification using quadric error metrics. *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '97*, 209–216. http://doi.org/10.1145/258734.258849

[6] Cignoni, P., Cignoni, P., Callieri, M., Callieri, M., Corsini, M., Corsini, M., … Ranzuglia, G. (2008). MeshLab: An Open-Source Mesh Processing Tool. *Sixth Eurographics Italian Chapter Conference*, 129–136. http://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136

[7] Botsch, M., Steinberg, S., Bischoff, S., & Kobbelt, L. (2002). OpenMesh – a generic and efficient polygon mesh data structure. *OpenSG Symposium*. Obtenido de http://www.graphics.rwth-aachen.de/media/papers/openmesh1.pdf

[8] Cignoni, P., Rocchini, C., & Scopigno, R. (1998). Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum*, *17*(2), 167–174. http://vcg.isti.cnr.it/publications/papers/metro.pdf

[9] libQGLViewer. (2015). Obtenido el 30 de marzo del 2016, de http://libqglviewer.com/

[10] Huttenlocher, D. P., Klanderman, G. A., & Rucklidge, W. J. (1993). Comparing Images Using the Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*(9), 850–863. http://doi.org/10.1109/34.232073

[11] Aspert, N., Santa-Cruz, D., & Ebrahimi, T. (2002). MESH: Measuring errors between surfaces using the Hausdorff distance. In *Proceedings - 2002 IEEE International Conference on Multimedia and Expo, ICME 2002* (Vol. 1, pp. 705–708). http://doi.org/10.1109/ICME.2002.1035879

[12] Akenine-Möllser, T. (2001). Fast 3D Triangle-Box Overlap Testing. *Journal of Graphics Tools*, *6*(1), 29–33. http://doi.org/10.1080/10867651.2001.10487535

[13] OpenMesh: The Halfedge Data Structure. Obtenido el 10 de octubre del 2017, de https://www.openmesh.org/Daily-Builds/Doc/a03967.html

[14] Foley, James D.; Andries van Dam; John F. Hughes; Steven K. Feiner (1990). "Spatial-partitioning representations; Surface detail". Computer Graphics: Principles and Practice. The Systems Programming Series. Addison-Wesley.

[15] SAT (Separating Axis Theorem). Obtenido el 10 de octubre del 2017, de http://www.dyn4j.org/2010/01/sat/

[16] 3D Printer Nozzle Comparison Guide. Obtenido el 10 de octubre del 2017, de https://www.matterhackers.com/news/3d-printer-nozzle-comparison-guide

10 GLOSARIO DE ABREVIATURAS

FFF (Fused Filament Fabrication): Fabricación mediante fundido de filamentos

3D: 3 Dimensiones

CAD (Computer-aided design): Diseño asistido por computador

11 ÍNDICE DE FIGURAS